

# Weekly Report

5/16/2016-5/22/2016

## Work

- Write a report about several machine learning algorithm including curve fitting, PCA, E-M algorithm, Levenberg-Marquardt method and SVM with quadratic programming.

## Plan for next week

- Report a paper on group meeting.
- Prepare to write a review.

## 1 方法概述

### 1.1 Curve fitting

给定从曲线上采样得到的点 $(x_i, y_i), i = 1, 2, \dots, N$ ，用多项式

$$y(x, \mathbf{w}) = w_0 + w_1x + w_2x^2 + \dots + w_Mx^M = \sum_{j=0}^M w_jx^j$$

来逼近原始曲线。我们通过降低误差平方和来求得多项式曲线的系数：

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N (y(x_n, \mathbf{w}) - y_n)^2$$

对参数求偏导，找到目标函数最低点：

$$\frac{\partial E}{\partial w_j} = \sum_{n=1}^N [(w_0 + w_1 x_n + w_2 x_n^2 + \dots + w_M x_n^M) - y_n] x_n^j = 0, j = 1, 2, \dots, M$$

等价于

$$w_0 \sum_{n=1}^N x_n^j + w_1 \sum_{n=1}^N x_n^{j+1} + \dots + w_M \sum_{n=1}^N x_n^{j+M} = \sum_{n=1}^N x_n^j y_n, j = 1, 2, \dots, M$$

也可以写成矩阵的形式

$$\begin{bmatrix} n & \sum_{n=1}^N x_n & \cdots & \sum_{n=1}^N x_n^M \\ \sum_{n=1}^N x_n & \sum_{n=1}^N x_n^2 & \cdots & \sum_{n=1}^N x_n^{M+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{n=1}^N x_n^M & \sum_{n=1}^N x_n^{M+1} & \cdots & \sum_{n=1}^N x_n^{2M} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_M \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

误差函数加入正则项：

$$\hat{E}(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N (y(x_n, \mathbf{w}) - y_n)^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

对参数求偏导：

$$\frac{\partial E}{\partial w_j} = \sum_{n=1}^N [(w_0 + w_1 x_n + w_2 x_n^2 + \dots + w_M x_n^M) - y_n] x_n^j + \lambda w_j = 0, j = 1, 2, \dots, M$$

写成矩阵的形式：

$$\begin{bmatrix} \lambda + n & \sum_{n=1}^N x_n & \cdots & \sum_{n=1}^N x_n^M \\ \sum_{n=1}^N x_n & \lambda + \sum_{n=1}^N x_n^2 & \cdots & \sum_{n=1}^N x_n^{M+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{n=1}^N x_n^M & \sum_{n=1}^N x_n^{M+1} & \cdots & \lambda + \sum_{n=1}^N x_n^{2M} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_M \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

即

$$(A^T A + \lambda E) \mathbf{w} = A^T \mathbf{y}$$

$$A = \begin{bmatrix} 1 & x_1 & \cdots & x_1^M \\ 1 & x_2 & \cdots & x_2^M \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \cdots & x_n^M \end{bmatrix}$$

## 1.2 PCA

PCA的目的是要对原始数据进行降维，同时尽可能保留数据的原始面貌。如果是降到二维，数据点 $\mathbf{y}$ 可以看做特征空间基的线性组合：

$$\mathbf{y} \approx \tilde{x} + w_1 \mathbf{x}_1 + w_2 \mathbf{x}_2, \mathbf{y}, \mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^{d \times 1}$$

所以我们要保证对所有数据点的误差和最小：

$$\underset{\mathbf{X}, \mathbf{W}}{\operatorname{argmin}} \|\mathbf{Y} - \mathbf{X}^T \mathbf{W}\|, \mathbf{Y} \in \mathbb{R}^{d \times N}, \mathbf{X} \in \mathbb{R}^{p \times d}, \mathbf{W} \in \mathbb{R}^{p \times N}$$

其中 $d$ 是数据维度， $p$ 是特征空间维度， $N$ 是样本数。首先我们把 $\mathbf{Y}$ 中心化： $\hat{\mathbf{Y}} = \mathbf{Y} - \bar{\mathbf{Y}}$ 。对 $\hat{\mathbf{Y}}$ 运用SVD：

$$\hat{\mathbf{Y}} = \mathbf{U} \mathbf{S} \mathbf{V}^T, \mathbf{U} \in \mathbb{R}^{d \times d}, \mathbf{V} \in \mathbb{R}^{N \times N}, \mathbf{S} \in \mathbb{R}^{d \times N}$$

那么 $\underset{\mathbf{X}, \mathbf{W}}{\operatorname{argmin}} \|\mathbf{Y} - \mathbf{X}^T \mathbf{W}\|$ 的解就是 $\mathbf{X} = \mathbf{U}^T, \mathbf{W} = \mathbf{S} \mathbf{V}^T$

## 1.3 EM

多元高斯模型具有以下形式：

$$N(\mathbf{x}|\mu, \Sigma) = \frac{1}{(2\pi)^{D/2} |\Sigma|^{1/2}} \exp \left\{ -\frac{1}{2} (\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu) \right\}$$

其中 $\mu$ 是 $D$ 维均值向量， $\Sigma$ 是 $D \times D$ 协方差矩阵， $|\Sigma|$ 表示 $\Sigma$ 的行列式。高斯混合模型可以表示为：

$$p(\mathbf{x}|\theta) = \sum_{i=1}^K p_i N(\mathbf{x}|\mu_i, \Sigma_i)$$

我们可以通过EM算法迭代求解系数：

Step1: 根据当前的 $p_k, \mu_k, \Sigma_k, k = 1, 2, \dots, K$ , 计算每个数据点属于各个多元高斯分布的概率 $Z_{nk}$ :

$$Z_{nk} = \frac{p_k N(x_n | \mu_k, \Sigma_k)}{\sum_{j=1}^K p_j N(x_n | \mu_j, \Sigma_j)}$$

Step2: 根据 $Z_{nk}$ 我们得到新的系数:

$$\begin{aligned} N_k &= \sum_{n=1}^N Z_{nk} \\ \mu_k &= \frac{1}{N} \sum_{n=1}^N Z_{nk} x_n \\ p_k &= \frac{N_k}{N} \\ \Sigma_k &= \frac{1}{N} \sum_{n=1}^N Z_{nk} (x_n - \mu_k)^T (x_n - \mu_k) \end{aligned}$$

## 1.4 Levenberg-Marquardt method

求解最优化问题

$$\min f(x), x \in \mathbb{R}^n$$

通常使用迭代搜索 $x_{k+1} = x_k + \alpha_k d_k$ ,  $\alpha_k$ 是步长,  $d_k$ 是搜索方向。可以使用最速下降法或者牛顿法求解。最速下降法是把负梯度方向最为搜索方向 $d_k = -g_k$ 。牛顿法是把当前点所在的局部信息看做是二次函数, 通过极小化二次函数达到在原函数上的下降。

$$f(x_k + s) \approx q^k(s) = f(x_k) + \nabla f(x_k)^T s + \frac{1}{2} s^T \nabla^2 f(x_k) s$$

$$x_{k+1} = x_k - [\nabla^2 f(x_k)]^{-1} \nabla f(x_k) = x_k - G_k^{-1} g_k$$

最速下降法在接近极小点时, 迭代较慢, 而牛顿法在接近极小点时, 收敛速度较快, 所以我们应当结合使用。所以使用Levenberg-Marquardt方法。

$$\min q^k(s) = f_k + g_k^T s + \frac{1}{2} s_k^T s, s.t. \|s\|_2 \leq h_k$$

引入拉格朗日函数 $L(s, \mu) = q^k(s) + \frac{1}{2} \mu (s^T s - h_k^2)$ , 要使 $(\nabla_s L = 0) g_k + (G_k + \mu_k I) s = 0, \mu \geq 0$

- 1) 给定初始点  $x_0, \mu_0 > 0, k = 1$
- 2) 计算  $g_k, G_k$
- 3) 若  $\|g_k\| < \epsilon$ , 停止
- 4) 尝试Cholesky分解  $G_k + \mu_k I$ , 若不能分解则不正定, 令  $\mu_k = 4\mu_k$ , 重复4直到正定
- 5) 求解  $(G_k + \mu_k I)s = -g_k$
- 6)  $r_k = \frac{f(x_k + s_k) - f(x_k)}{q^k(s_k) - q^k(0)}$
- 7) 若  $r_k \leq 0.25$ , 置  $\mu_{k+1} = 4\mu_k$ ; 若  $r_k \geq 0.75$ , 置  $\mu_{k+1} = \mu_k/2$ ;
- 8) 若  $r_k \leq 0$ , 置  $x_{k+1} = x_k$ , 否则  $x_{k+1} = x_k + s_k$
- 9) 令  $k = k + 1$ , 转2

## 1.5 SVM

对于训练样本集  $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}, y_i \in \{-1, +1\}$ , 我们希望有一个划分超平面  $\mathbf{w}^T \mathbf{x} + b = 0$  把两类训练样本分开, 即

$$\mathbf{w}^T \mathbf{x}_i + b \geq +1, y_i = +1$$

$$\mathbf{w}^T \mathbf{x}_i + b \leq -1, y_i = -1$$

样本空间中任意点到超平面的距离可以写为:

$$r = \frac{|\mathbf{w}^T \mathbf{x} + b|}{\|\mathbf{w}\|}$$

使得等式  $\mathbf{w}^T \mathbf{x} + b = \pm 1$  成立的数据点是支持向量, 两类支持向量到超平面的和成为最大间隔:

$$\gamma = \frac{2}{\|\mathbf{w}\|}$$

我们的目标是求解一下目标函数:

$$\begin{aligned} \max_{\mathbf{w}, b} \quad & \frac{2}{\|\mathbf{w}\|} \\ \text{s.t.} \quad & y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1, i = 1, 2, \dots, m \end{aligned}$$

上述问题可以使用拉格朗日乘子法得到对偶问题:

$$\begin{aligned} \min_{\alpha} \quad & \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j - \sum_{i=1}^m \alpha_i \\ \text{s.t.} \quad & \sum_{i=1}^m \alpha_i y_i = 0, \alpha_i \geq 0, i = 1, 2, \dots, m \end{aligned}$$

这是一个有等式约束和不等式约束的二次规划问题。对于标准的二次规划问题：

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \phi(x) &= \frac{1}{2} x^T H x + c^T x \\ \text{subject to } & A x = b, D x \geq f \end{aligned}$$

其中 $H$ 是对称的，约束矩阵 $A$ 和 $D$ 是 $m \times n$ 和 $m_D \times n$ 的。

首先只考虑等式约束的条件下，利用拉格朗日乘子法，就只需要求解 $Hx - A^T \lambda = -c$ 和 $Ax = b$ ，即

$$\begin{pmatrix} H & A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} x \\ \lambda \end{pmatrix} = \begin{pmatrix} -c \\ b \end{pmatrix}$$

其中 $K = \begin{pmatrix} H & A^T \\ A & 0 \end{pmatrix}$ 是KKT矩阵。

对于不等式约束可以使用有效集方法。基本思想是，约束条件缩小了可行解可以取值的区域，约束条件可以看做是区域的边界，如果最优解处于某个边界上，那么这个边界可以直接看做等式条件，否则不处于某个边界上的话，说明这个约束条件不是必须的。基于这样的想法，我们每次在一定的有效集下求解等式约束最优解，当最优解要超过边界时，把这个边界考虑进来进一步求解等式约束的最优解。

1) 确定初始点 $x_0$ ，以及对应的约束构成的执行集 $W$

2) 在执行集 $W$ 的情况下，求解目标方程：

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \phi(x) &= \frac{1}{2} x^T H x + c^T x \\ \text{subject to } & A x = b, D_w x = f_w \end{aligned}$$

其中 $D_w$ 和 $f_w$ 都是执行集 $W$ 中的不等式约束对应的那几行。这就是一个等式约束问题，只要求解

$$\begin{pmatrix} H & A_w^T \\ A_w & 0 \end{pmatrix} \begin{pmatrix} x \\ \lambda_w \end{pmatrix} = \begin{pmatrix} -c \\ b_w \end{pmatrix}$$

其中 $A_w = \begin{pmatrix} A \\ D_w \end{pmatrix}$ ,  $\lambda_w$ 是 $\begin{pmatrix} \lambda \\ \lambda_{D_w} \end{pmatrix}$ ,  $b_w = \begin{pmatrix} b \\ f_w \end{pmatrix}$

若 $x = x_k$ 即，以及到达最优解，转3)，否则转4)

3) 如果 $\lambda_{D_w}$ 的所有值都大于等于0，则算法停止，否则

$$i_k = \underset{i \in W_k}{\operatorname{argmin}} (\lambda_{D_w})_i, x_{k+1} = x, W_{k+1} = W_k \setminus \{i_k\}$$

4)  $p_k = x - x_k, \alpha_k = \min\{1, \min_{i \notin W_k, D_i^T p_k < 0} \frac{f_i - D_i^T x_k}{D_i^T p_k}\}$ 。如果  $\alpha_k = 1$  则表明其他不等式约束没有对解造成影响，若果  $\alpha_k < 1$  说明碰到了不等式约束的边界，这时我们只能要保证所有约束仍然是成立的，即走满足所有条件下的最大步长，这时达到了某些不等式约束，考虑使得等式成立， $W_{k+1} = W_k \cup i_k$ ，否则执行集不变。转2)。

## 2 实验结果

### 2.1 Curve fitting

多项式曲线拟合的结果如下，对  $\sin(x)$ （图中蓝线）进行有噪声的采样（圆点），最后把拟合的结果函数绘制出来绿色。S是采样的数量，M的多项式阶数。  
(a-d) 都是对  $\sin(x)$  采样了10个点，随着阶数调高，曲线越来越接近  $\sin(x)$ ，但阶数过高时会产生过拟合现象，如图(d)。消除过拟合现象可以通过增加采样点(e)，或者在求参数时加入正则项(f)。

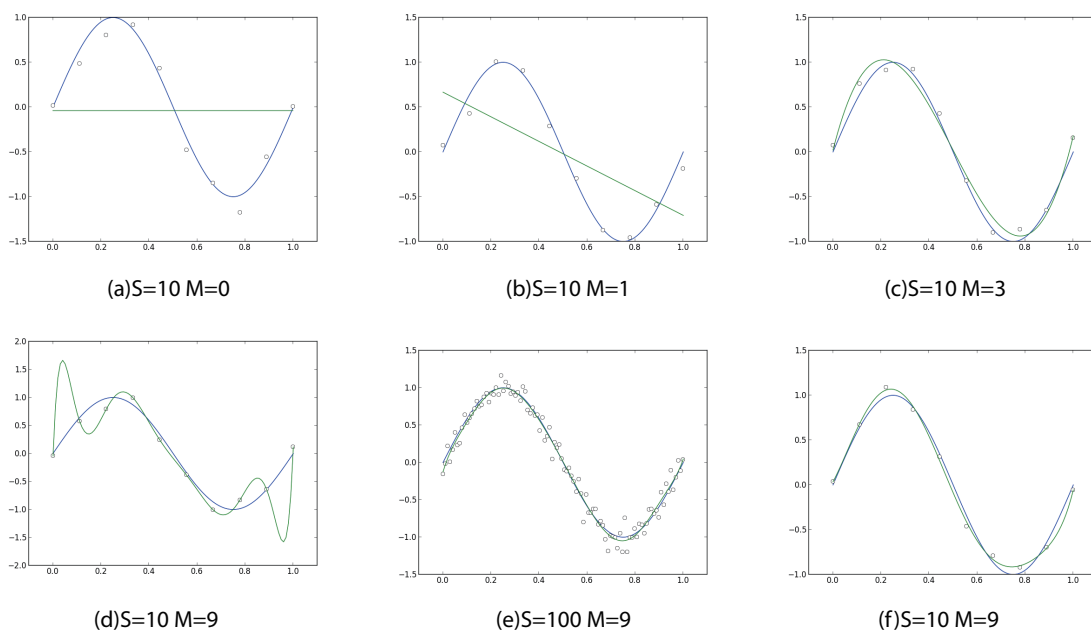


Figure 1: 曲线拟合结果

## 2.2 PCA

如图2所示，199个32\*32的数字为3的图像被PCA降维到二维平面上。然后在二维平面上打一个5\*5的方格，里面随机采样一个点，显示在右边。

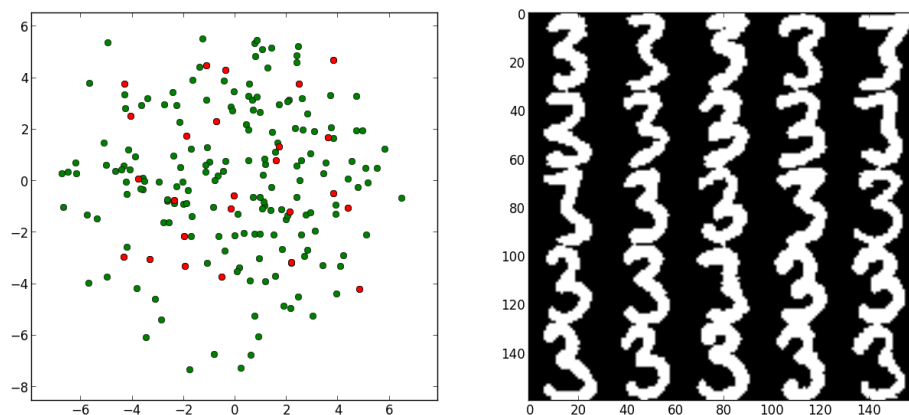


Figure 2: PCA结果

## 2.3 E-M method

如图3所示，两堆点是两个二维高斯分布的采样，两个曲线是EM算法求得的原高斯分布的等值曲线。表2.3显示的是混合高斯分布参数以及EM算法求得的参数。

概率	均值	协方差矩阵	求解概率	求解均值	求解协方差矩阵
0.7	$\begin{pmatrix} 2 \\ 3 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 \\ 1 & 3 \end{pmatrix}$	0.67007738	$\begin{pmatrix} 2.07484958 \\ 3.04829278 \end{pmatrix}$	$\begin{pmatrix} 1.04060236 & 1.04178103 \\ 1.04178103 & 3.44353391 \end{pmatrix}$
0.3	$\begin{pmatrix} -5 \\ -4 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 \\ 1 & 3 \end{pmatrix}$	0.32992262	$\begin{pmatrix} -4.98480768 \\ -3.83038073 \end{pmatrix}$	$\begin{pmatrix} 1.0807762 & 0.97190332 \\ 0.97190332 & 2.82805083 \end{pmatrix}$

Table 1: 混合高斯分布参数以及EM算法求得的参数



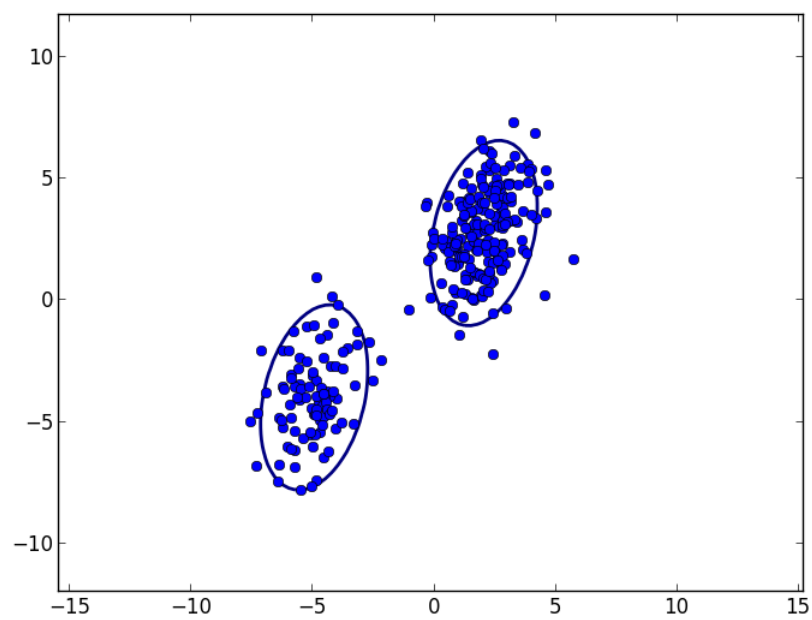


Figure 3: EM算法结果

## 2.4 Levenberg-Marquardt method

以优化一下函数为例：

$$f(x) = \sum_{i=1}^{N-1} 100(x_i - x_{i-1}^2)^2 + (1 - x_{i-1})^2$$

我们求得它的梯度：

$$\begin{aligned}\frac{\partial f}{\partial x_j} &= 200(x_j - x_{j-1}^2) - 400x_j(x_{j+1} - x_j^2) - 2(1 - x_j) \\ \frac{\partial f}{\partial x_0} &= -400x_0(x_1 - x_0^2) - 2(1 - x_0) \\ \frac{\partial f}{\partial x_j} &= 200(x_{N-1} - x_{N-2}^2)\end{aligned}$$

Hessian矩阵为：

$$\begin{aligned}\frac{\partial^2 f}{\partial x_0^2} &= 1200x_0^2 - 400x_1 + 2 \\ \frac{\partial^2 f}{\partial x_0 \partial x_1} &= \frac{\partial^2 f}{\partial x_1 \partial x_0} = -400x_0 \\ \frac{\partial^2 f}{\partial x_{N-1} \partial x_{N-2}} &= \frac{\partial^2 f}{\partial x_{N-2} \partial x_{N-1}} = -400x_{N-2} \\ \frac{\partial^2 f}{\partial x_{N-1}^2} &= 200 \\ H_{i,j} &= \frac{\partial^2 f}{\partial x_i \partial x_j} = (202 + 1200x_i^2 - 400x_{i+1})\delta_{i,j} - 400x_i\delta_{i+1,j} - 400x_{i-1}\delta_{i-1,j}\end{aligned}$$

如图4给定初值 $(-10, -10, -10, -10)$ 最终能收敛到结果 $(1, 1, 1, 1)$ 。

## 2.5 SVM

下面的例子以数据集 $D = \{(2, 6, +1), (2, 4, -1), (4, 2, -1), (4, 4, +1)\}$ 为例。

```

/System/Library/Frameworks/Python.framework/Versions/2.6/bin/python2.6 /Users/zhuminfeng/Documents/Project/Applied-Mathematics-for-Computer-Science/hw4.py
[ -6.56772106 -6.39772142 -9.78823053 95.07921449]
[ -4.27606999 -4.24656828 -2.29893977 -50.4771801 ]
[ -2.76111285 -2.64997484 -2.22487066 4.80747269]
[ -2.76111285 -2.64997484 -2.22487066 4.80747269]
[ -2.76111285 -2.64997484 -2.22487066 4.80747269]
[ -1.73549295 -1.8549059 0.75629772 -0.05806787]
[ -1.0582207 -1.23611345 0.76367462 0.41370897]
[ -0.62187299 -0.76952416 0.48191565 0.15543729]
[ -0.4429766 -0.15751554 -0.22284362 -0.44402902]
[ -0.34410024 0.04456186 -0.12447629 0.0046964 ]
[ -0.27225181 0.07189649 0.01120304 -0.01825468]
[ -0.27225181 0.07189649 0.01120304 -0.01825468]
[ -0.27225181 0.07189649 0.01120304 -0.01825468]
[ -0.27225181 0.07189649 0.01120304 -0.01825468]
[ -0.01495878 -0.05416347 -0.00230666 -0.00035618]
[ 4.78416211e-02 4.31091461e-03 6.19140676e-03 -4.62794398e-05]
[ 4.78416211e-02 4.31091461e-03 6.19140676e-03 -4.62794398e-05]
[ 1.68223519e-01 2.24393750e-02 9.78850443e-03 7.33806593e-05]
[ 3.46048530e-01 9.60266146e-02 1.35212658e-02 1.65214872e-04]
[ 0.5001688 0.23421305 0.04468848 0.00100879]
[ 0.66565543 0.41930247 0.14881219 0.01120130]
[ 0.73590886 0.53686643 0.27807483 0.06037003]
[ 0.82027147 0.67842202 0.43978461 0.16699380]
[ 0.85747848 0.73395003 0.53498121 0.27700499]
[ 0.92539398 0.85143564 0.71056227 0.47394695]
[ 0.93346354 0.87097517 0.75761112 0.57169991]
[ 0.93346354 0.87097517 0.75761112 0.57169991]
[ 0.97594997 0.95059013 0.897099 0.78506304]
[ 0.97920333 0.95873042 0.91889025 0.843811 ]
[ 0.99543042 0.99060206 0.98023859 0.9570333 ]
[ 0.99787824 0.99574541 0.99146244 0.98286373]
[ 0.99983801 0.99967158 0.99932638 0.99858891]
[ 0.99999309 0.99998612 0.99997207 0.99994362]
[ 0.99999989 0.99999978 0.99999956 0.99999912]
[ 1. 1. 1. 0.99999999]
[ 1. 1. 1. 0.99999999]

```

Figure 4: LM算法收敛过程

## References

- [1] 李航. 统计学习方法[M]. 清华大学出版社, 2012.
- [2] 周志华. 机器学习. 清华大学出版社, 2016.
- [3] Wong E L S. Active-Set Methods for Quadratic Programming[J]. Dissertations Theses - Gradworks, 2011, 126(2):857-893.
- [4] Madsen K, Nielsen H B, Tingleff O. Methods for Non-Linear Least Squares Problems (2nd ed.)[J]. Siam J Optim, 2004, 16(1):487 – 490.

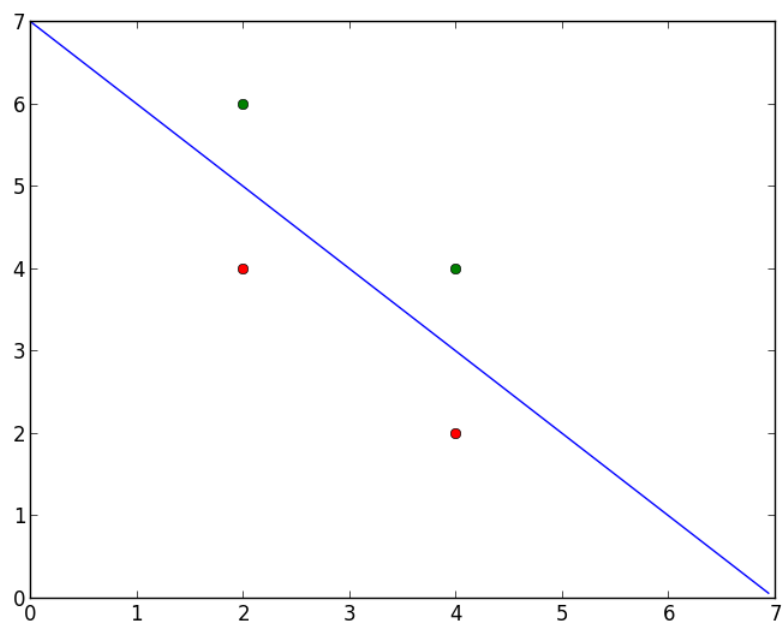


Figure 5: SVM简单例子